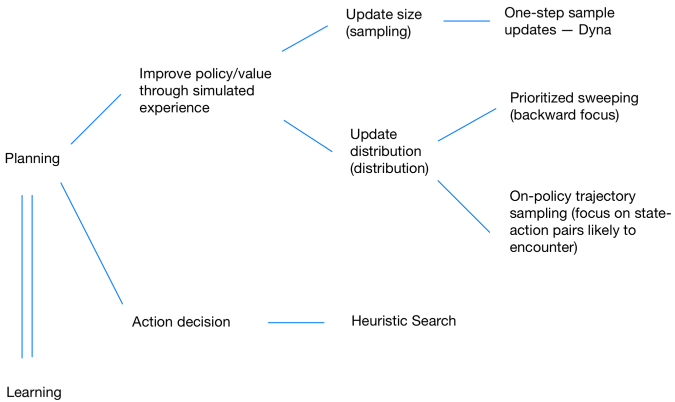
Reinforcement learning: An introduction
Chapter 8

RL Reading Group

Chapter 8: Planning and Learning with Tabular Methods



Introduction to Planning and Learning

- **Models**: Models: distribution models and sample models
- **Planning**: Takes a model as input and produces or improves a policy for interacting with the modeled environment
- **Learning**: planning uses simulated experience generated by a model, learning methods use real experience generated by the environment

$$\textit{Model} \longrightarrow \text{Simulated experience} \xrightarrow{\text{backups}} \textit{Values} \longrightarrow \textit{Policy}$$

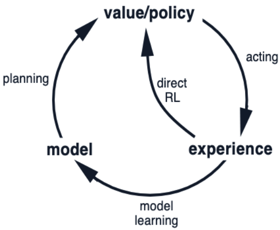
Planning vs. Learning

Random-sample one-step tabular Q-planning

Loop forever:
1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

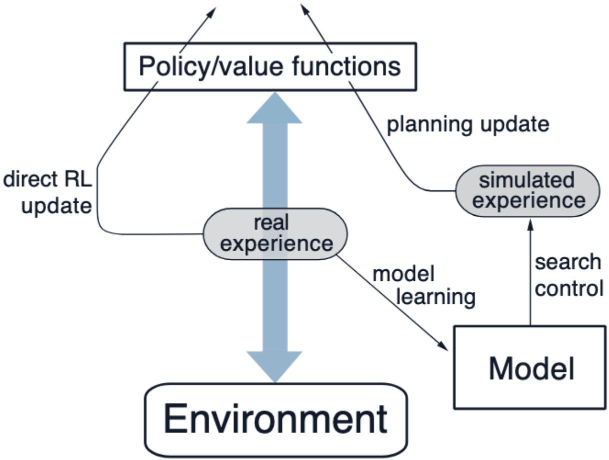
Figure: Planning based on 1-step Q-leaning from sample model

Planning, Acting, and Learning



- Model-learning
  - Pros: achieve a better policy with fewer environmental interactions
- Direct Reinforcement-learning(previous chapters)
  - Pros: not affected by biases in the design of the model
- Recognize the similarity between two sides than by opposing them
  - Example: Dynamic Programming(planning) and TD methods(model-free learning)

Dyna: Integrated Planning, Acting, and Learning



Dyna-Q

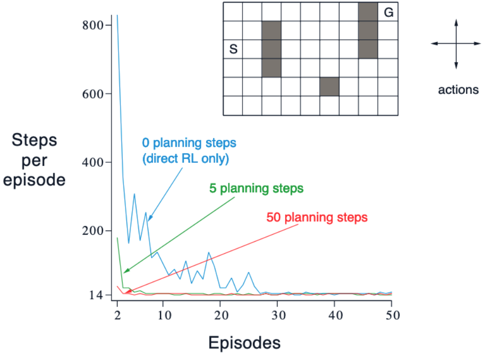
- Dyna-Q is a combination of planning and learning:
  1. Direct RL: Update value function based on real experience.
  2. Indirect RL: Update value function based on simulated experience from the model.
- The Dyna-Q algorithm:
  1. Take action A_t, observe R_{t+1}, S_{t+1}.
  2. Update $Q(S_t, A_t)$.
  3. Model update: $Model(S_t, A_t) \leftarrow (R_{t+1}, S_{t+1})$.
  4. Repeat n times:
     - 4.1 Sample (S, A) from previously observed states and actions.
     - 4.2 Simulate next state and reward: $(R, S') \leftarrow Model(S, A)$.
     - 4.3 Update $Q(S, A)$.

Dyna-Q Algorithm

Tabular Dyna-Q

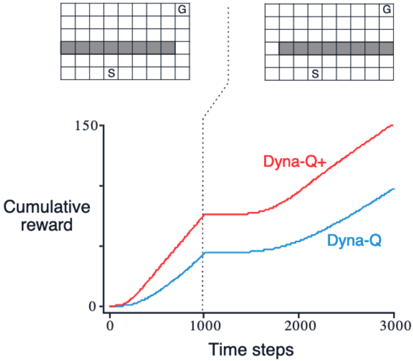
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S, Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Example: Dyna-Maze Multi-step bootstrap?

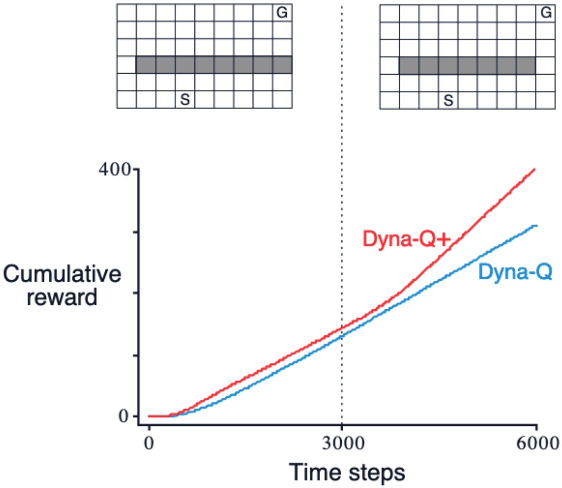


- 47 states, 4 actions
- reward zero on all transitions, +1 on reaching the goal
- reached the goal and return to start with discount factor $\gamma = 0.95$

Example: Dyna-Maze (Model is Wrong)



Example: Dyna-Maze Continued



- Another version of exploration-exploitation conflict!

Prioritized sweeping

- Much more efficient if simulated transitions and updates are focused on particular state-action pairs
- A queue is maintained of every state–action pair whose estimated value would change nontrivially if updated

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Loop forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow policy(S, Q)$
  (c) Take action A; observe resultant reward, R, and state, S'
  (d) $Model(S, A) \leftarrow R, S'$
  (e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$.
  (f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
  (g) Loop repeat n times, while $PQueue$ is not empty:
    $S, A \leftarrow first(PQueue)$
    $R, S' \leftarrow Model(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    Loop for all $\bar{S}, \bar{A}$ predicted to lead to S:
      $\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
      $P \leftarrow |\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})|$.
      if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

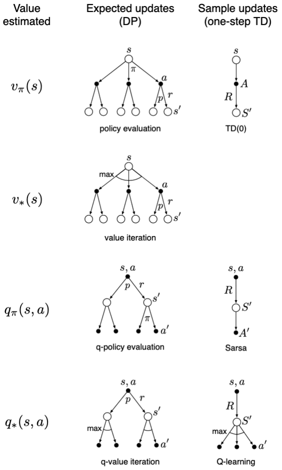
Prioritized Sweeping Cons

- Use expected updates, and waste lots of computation on low-probability transitions.
- Focus of one-step updates in this book is along three dimensions:

$$\begin{cases} 1 & \text{if they update state values or action values} \\ 2 & \text{if they estimate the value for the optimal policy} \\ & \text{or for an arbitrary given policy} \\ 3 & \text{if the updates are expected updates, considering all possible} \\ & \text{events that might happen, or sample updates, considering} \\ & \text{a single sample of what might happen} \end{cases}$$

"Hyper-Generalization": Expected vs. Sample Updates

- Four classes of updates for approximating the four value functions: q^*, v^*, q^π, and v^π.



Expected vs. Sample Updates for approximating q_*

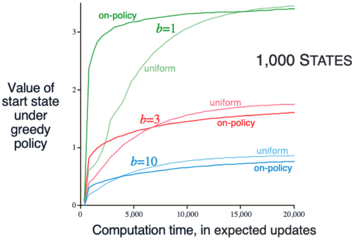
- Expected Update:
  $Q(s, a) \leftarrow \sum_{s',r} \hat{p}(s', r \mid s, a) \left[r + \max_{a'} Q(s', a')\right]$
- Sample Update (Q-learning-like Update):
  $Q(s, a) \leftarrow Q(s, a) + \alpha \left[R + \max_{a'} Q(S', a') - Q(s, a)\right]$

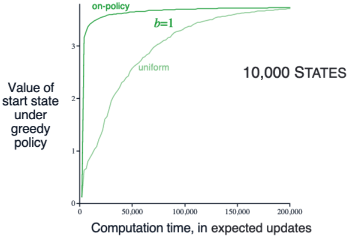
Distributing Updates

- Method 1. Perform sweeps through the entire state-action pairs
- Method 2. Sample from state-action pairs according to some distribution
  (i.e. one simulates explicit individual trajectories and performs updates at the state or state-action pairs encountered along the way.)

Trajectory Sampling

- Short term: on-policy helps by focusing on descendants of the start state
- Long term: on-policy may hurt: commonly occuring states have correct values





Decision Time Planning

Two ways to think about planning:

- Gradually improve a policy or value function on the basis of simulated experience obtained for a model
- A computation to select a single action A_t at a new state S_t
  - values and policy created by the planning process are discarded after being used to select the current action.

Heuristic Search

- Intuition: focus on computation and memory resources on the current decision
  - Example: chess has far too many possible positions to store distinct value estimates for each of them, but can store a distinct estimate for millions of forward-looking positions from a single postition
- Performance improvement observed with deeper search not due to multistep updates, but due to the focus and concentration of updates on states and actions immediately downstream from the current state

Example: Heuristic Search

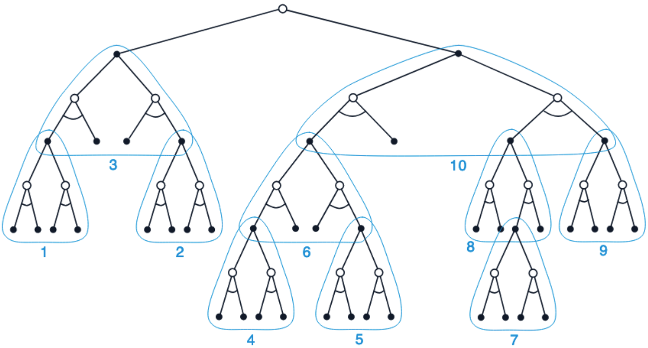


Figure 8.9: Heuristic search can be implemented as a sequence of one-step updates (shown here outlined in blue) backing up values from the leaf nodes toward the root. The ordering shown here is for a selective depth-first search.